THERE IS CLAIMED:

1. A method of verification for a design, comprising:

providing a description of said design;

specifying correctness criteria for said design, wherein said correctness criteria are expressed as one or more correctness properties;

abstracting said design description to provide an abstract model of said design;

generating a witness graph for said one or more correctness properties based on a deterministic analysis of said abstract model;

determining a conclusive result from the set consisting of property violation and property satisfaction, when said witness graph is empty; and

generating a testbench automatically when said witness graph is not empty, and

performing simulation with said testbench;

wherein, when a property refers to universal path quantification, said witness graph includes paths demonstrating only said property violation, defining counter-examples;

wherein, when said property refers to existential path quantification, said witness graph includes paths demonstrating only said property satisfaction, defining witnesses;

wherein said conclusive result is said property satisfaction when said property refers to said universal path quantification; and

wherein said conclusive result is said property violation when said property refers to said existential path quantification.

2. The method for verification as set forth in claim 1, wherein:

said generation of said testbench comprises:

determining embedded constraints for guiding vector generation based on said witness graph;

determining priorities for guiding said vector generation based on said witness graph;

generating a vector generator module including said embedded constraints and said priorities; and

generating a monitor module, said monitor module checking said conclusive result;

wherein, when said property refers to said universal path quantification, said vector generator module is generated so that said generated vectors are directed toward finding said counter-examples, and

wherein, when said property refers to said existential path quantification, said vector generator module is generated so that said generated vectors are directed toward finding said witnesses; and

said simulation of said design, using said generated test bench, comprises;

generating said vectors with said vector generator module based on said embedded constraints, including generating random patterns and using said constraints as a filter to select desirable ones of said random patterns; and

checking said monitor module for property violation or satisfaction.

3. The method of verification as set forth in claim 2, wherein:

said embedded constraints are derived from transition conditions in said witness graph; and

said priorities are associated with transitions in said witness graph.

4. The method of verification as set forth in claim 3, wherein:

said priorities are generated from said witness graph based on one or more of:

distance to targets,

transition probabilities, and

simulator trace data.

5. The method of verification as set forth in claim 1, wherein:

said generation of said witness graph comprises:

removing a portion from said design when an influence determination does not indicate that said portion of said design is in a cone of influence of said property;

modeling, as an initial abstract model, a controller state and variables in a datapath state directly involved in predicates of said correctness property;

performing deterministic analysis on said abstract model; and

pruning said abstract model to obtain said witness graph;

said influence determination indicates said portion of said design is in said cone of influence of said property when said portion of said design is one or more of:

a portion directly affecting said variables in said predicates of said property, and

a portion affecting branching which in turn affects predicates of said property;

said deterministic analysis determines which portion in said abstract model indicates paths relating to said conclusive result for said property;

said pruning comprises removing a portion in said abstract model indicated by said analysis not to relate to said conclusive result for said property.

6. The method of verification as set forth in claim 5, wherein:

said pruning is followed by a step of refining said abstract model by adding variables from said datapath state to provide a refined abstract model;

said analysis, pruning, and refining steps are performed in an iterative process; and

said witness graph is said refined abstract model at the end of said iterative process.

7. The method of verification as set forth in claim 5, wherein said property is represented using a computation tree logic (CTL) formula.

8. The method of verification as set forth in claim 7, wherein said step of analysis is performed by:

determining CTL subformulas of said CTL formula;

with each of said CTL subformulas, associating a given set of abstract states corresponding to an over-approximate set of concrete states satisfying said CTL subformula, said given set of abstract states defining an upper set;

for ones of said CTL subformulas beginning with an E-type operator, performing standard model checking over said abstract model;

for ones of said CTL subformulas beginning with an A-type operator:

selecting an E-type operator corresponding to said A-type operator and guaranteed to result in an over-approximation, and

computing an other set of abstract states, corresponding to an intersection of said upper set with a set recursively computed for the negation of said A-type operator, said other set of abstract states defining a negative set;

checking an initial state of the design to determine a conclusive result, wherein:

when said initial state does not belong to said upper set, determining said property to be conclusively proved to be false;

when said property represented using said CTL formula starts with said A-type operator, and when said initial state belongs to said upper set, and when said initial state does not belong to said negative set, determining said property to be conclusively proved to be true; and

determining said analysis to be inconclusive when said property is not conclusively to be proved to be one of true and false.

9. The method of verification as set forth in claim 8, wherein said step of pruning comprises:

marking witness states; and then

pruning unmarked states by replacement with a sink state having every transition therefrom leading to said sink state, and all atomic propositions in said sink state being assumed false.

10. The method of verification as set forth in claim 9, wherein said step of marking said witness states comprises:

computing a witness-top set of states consisting of the intersection of set of states reachable from initial state of said design and said upper set;

marking all of said witness-top set of states;

using a marking procedure, for each of said CTL subformulas of said CTL formula representing said property, with said witness-top set defining a care set, comprising the steps of:

associating with said CTL subformula, as a witness set thereof, a given set of states defined by the intersection of said upper set associated with said CTL subformula and said care set;

for ones of said CTL subformulas beginning with said EX operator:

marking additional states in the image of said witness set; and

recursively applying said marking procedure to the CTL subformulas thereof, beginning with said EX operator, with said additional states as said care set;

for ones of said CTL subformulas beginning with an A-type operator:

determining a neg-witness set as the intersection of said negative set associated with said A-type subformula and said care set; and

recursively applying said marking procedure on the negation of said A-type subformula, with said neg-witness set as said care set; and

for all other types of said CTL subformulas, applying said marking procedure recursively on the CTL subformulas thereof, with said witness set as said care set.

11. The method of verification as set forth in claim 10, wherein said vector generator module is generated to include a search of said witness and neg-witness sets of states for a concrete witness, returning an indication of success when finding said concrete witness.

12. The method of verification as set forth in claim 11, wherein said search is conducted using a backtracking method comprising:

specifying a given CTL formula;

specifying a given concrete state belonging to said witness set associated with said CTL formula;

starting from said concrete state;

determining an indication of success when there exists a concrete witness for said CTL formula, and failure otherwise; and

backtracking when said indication is failure, wherein:

when said CTL operator is not an A-type operator, search subproblems are conducted on the subformulas of said CTL subformula and each concrete state belonging to the associated witness sets;

when said CTL operator is an A-type operator and said state does not belong to said negative set, said indication is success;

when said CTL operator is an A-type operator and when said state belongs to said negative set, a search subproblem is set up with the negation of said CTL formula and said concrete state, wherein success of said negated subproblem indicates failure, and failure of said negated subproblem indicates success.

13. The method of verification as set forth in claim 12, wherein:

said search subproblems on said CTL subformulas and said concrete states belonging to the associated witness sets are set up in a prioritized manner based on one or more of:

distance to targets,

transition probabilities, and

simulator trace data.

14. A test bench generation apparatus, comprising:

an abstract control data flow graph (CDFG) generator, a witness graph generator, and a final stage module; said witness graph generator comprising a model checking interface, a model checker, an error trace generator, and a model iterator; said final stage module comprising a priority generator, and a test bench generator, and a simulator;

said abstract CDFG generator taking, as inputs, a parse tree of a computation tree logic (CTL) property and a database representing a CDFG, said CDFG being produced

from parsing a hardware description language description of a design, said CDFG representing a control part and a data part of said design;

said abstract CDFG generator generating from said inputs an abstract CDFG;

said model iterator receiving, as input, said abstract CDFG and performing a first iteration including constraint solving to prune paths, and producing a Level 1 model after said first iteration;

said model checking input interface transforming said Level 1 model to a form acceptable to said model checker;

said error trace generator identifying all error traces produced by said model checker, and capturing said traces in finite state machine (FSM) form, wherein said FSM form is a Level 2 model;

said Level 2 model constituting a final witness graph when said model checker provides a resource exhaustion indication;

said model iterator performing a subsequent iteration when said model checker provides an inconclusive indication with respect to said Level 2 model;

said priority generator assigning to each transition in said witness graph a respective priority representing a likelihood of that transition being a path segment of a counterexample or a witness, and being based on an evaluation of one or more of: the ease with which said transition can be taken, a number of paths following said transitions in said witness graph leading to a target state, and a distance of said transition from final states of said witness graph;

said test bench generator producing software code instructions comprising a test bench, and producing a database to be used by said test bench;

said test bench including instructions for guiding and directing said simulator by generating vectors, based on information from said database, until one of said paths in said witness graph has been completely simulated. ? correctness result?

15. An automatic test bench generation method for a hardware design, said hardware design being described in a hardware description and including correctness criteria expressed as a correctness property, said automatic test bench generation method comprising:

generating a witness graph based on said hardware description;

determining, based on said witness graph, embedded constraints for guiding vector generation;

generating a vector generator module including said embedded constraints; and

generating, based on said correctness criteria, a monitor module for checking a correctness result with respect to said correctness property.

16. A method for assessing simulation coverage of a given set of simulation vectors for a given design, comprising:

providing a description of said design;

specifying correctness criteria for said design, wherein said correctness criteria are expressed as one or more correctness properties;

generating a witness graph for one or more of said correctness properties; and

determining coverage of said witness graph, using said given set of simulation vectors, by marking entities visited by said given set of simulation vectors in said witness

graph, said entities being selected from the set consisting of states, transitions, and paths.

17. The method of assessing simulation coverage as set forth in claim 16, wherein:

said generation of said witness graph comprises:

removing a portion from said design when an influence determination does not indicate that said portion of said design is in a cone of influence of said property;

modeling, as an initial abstract model, a controller state and variables in a datapath state directly involved in predicates of said correctness property;

performing deterministic analysis on said abstract model; and

pruning said abstract model to obtain said witness graph;

said influence determination indicates said portion of said design is in said cone of influence of said property when said portion of said design is one or more of:

a portion directly affecting said variables in said predicates of said property, and

a portion affecting branching which in turn affects predicates of said property;

said deterministic analysis determines which portion in said abstract model indicates paths relating to said conclusive result for said property; and

said pruning comprises removing a portion in said abstract model indicated by said analysis not to relate to said conclusive result for said property.

18. The method of assessing simulation coverage as set forth in claim 16, wherein:

said pruning is followed by a step of refining said abstract model by adding variables from said datapath state to provide a refined abstract model;

said analysis, pruning, and refining steps are performed in an iterative process; and

said witness graph is said refined abstract model at the end of said iterative process.